Libera Università Internazionale
degli Studi Sociali Guido Carli

PREMIO TESI D'ECCELLENZA

Deep Learning for Asset Managers: Drivers that Move Ferrari on Wall Street

Linda Gentili

2020-2021

LUISS

Libera Università Internazionale
degli Studi Sociali Guido Carli

Working Papaer n. 4/2020-2021
Publication date: January 2023
Deep Learning for Asset Managers: Drivers that Move Ferrari on Wall Street

ISBN 978-88-6105-935-1

Luiss Academy is an imprint of
Luiss University Press – Pola Srl
Viale Pola 12, 00198 Roma
Tel. 06 85225485
E-mail lup@luiss.it
www.luissuniversitypress.it

Deep Learning for Asset Managers: Drivers that Move Ferrari on Wall Street

Linda Gentili

ABSTRACT

In this paper, we present a new approach to time series forecasting based on novel machine learning models. Time series forecasting is an important task for both traditional statistics and modern deep learning, with the first having held the leading role in the forecasting field for a long time and the second having just started to systematically outperform the statistical models' state-of-the-art.

The approach we employ addresses the forecasting task as a stock-price forecasting problem and compares the performances obtained with the traditional ARIMA model to those obtained with the sequential step-by-step modelling of the LSTM and the novel attention-based modelling of the Transformer. We proceed with an in-depth study of the Transformer neural networks by comparing the performances of the original Transformer architecture, which has established itself as the state-of-the-art in all natural language processing tasks, with those of the novel Temporal Fusion Transformer, introduced by Google in the 2019 to specifically apply the attention-based model to complex, high frequency and highly irregular time series frameworks where the temporal pattern is a core component to model.

1. INTRODUCTION

The frameworks for time series forecasting always require balancing the accurateness of the predictions with the robustness of good-generalizing forecasting models: a model that perfectly fits an identified pattern among variables must be discarded if it fails to forecast the time series when there's a change in the equilibrium of the environment; on the opposite, a model that doesn't just follow an initiated trend among variables but prevents a shift to a different equilibrium must be chosen for the pursed task as it meets the criteria for being a *universal approximator*: its accurateness and robustness derive from the fact that it identifies dynamical relations among variables and understands of how those relations change with a changing environment.

Both the academic community and the professionals of the financial industry have always been interested in forecasting financial time series, but with no surprise they've encountered major problems when applying the forecasting frameworks to the highly volatile and non-stationary nature of the stock market. In more recent times, the financial forecasting task has been addressed with a completely renewed interest and the drivers behind that switch have been summarized in two main factors: first, the

enormous quantity and the different types of information we have at disposal to feed any forecasting models; second, but not less important, the computational capabilities of the technology we rely upon to train increasingly complex forecasting models.

A well-known example of this paradigm is the Medallion Fund, a hedge fund founded by Jim Simons in 1988 which has generated average annual returns of 66% since first started, setting the record of 5 times greater returns than those of the S&P 500 but with a similar standard deviation (18.7% vs 17.0%, respectively). As clearly explained by Gregory Zuckerman in his best-selling book "*The man who solved the market*"[1], the reasons behind those performances are mainly linked to the leverage built upon the two factors named above—data and technology — both used through sophisticated mathematical and deep learning models to forecast the trends in the financial market and in its sentiment.

Without entering the details of the Medallion Fund trading models, we just bring back the words used by J. Hull in its "*Machine Learning in Business*"[2] book to describe Jim Simons' approach. Those words can effectively summarize the reasons that moved the interest behind this work:

> "New data sources are becoming available all the time. One approach is to try and be one step ahead of most others in exploiting these new data sources. Another one is to develop better models than those being used by other and then be very secretive about it. Renaissance Technology provides an example of the second approach. It has been very successful at using sophisticated models to understand stock price patterns."

Building upon those premises, this work applies current state-of-the-art forecasting models to predict the stock price of Ferrari, the Italian luxury car manufacturer founded by Enzo Ferrari in 1947 in Maranello and quoted on the NYSE in 2015 and on Borsa di Milano in 2016.

The paper is organized as follows: Section II explores the background of this work and the related works, highlighting the models that are considered state-of-the-art in time series forecasting as of today. Section III addresses the theory behind this work, with a particular focus given to the models developed on later sections. Section IV sets the mathematical formulation of this work's forecasting task. Section V presents the methodology, with its sub-sections describing in detail each step of the work. Section VI presents the analysis of the results. Section VII summarizes the paper's content and draws the conclusion and future possible developments.

1. G. Zuckerman, *The man who solved the market: how Jim Simons launched the quant revolution*, Penguin Putnam Inc, 2019.
2. J. Hull, *Machine Learning in Business*, Independently published, 2019.

2. BACKGROUND AND RELATED WORKS

According to the Efficient Market Hypothesis (EMH)[3][4], a financial time series is nearly unforecastable because prices reflect all relevant available information and, as soon as new information come out, that information is immediately reflected into prices by market participants seeking arbitrage opportunities. The conclusion is that there's no chance to *beat the market*, meaning that market participant cannot earn above average returns without taking additional risks. However, the empirical evidence has questioned the EMH's assumptions through times, showing that at least two out of the three main assumptions behind the EMH are unrealistic:

- investors act as rational agents, aiming at maximizing their profits;
- all market participants share the same information at the same time.

We leave out of our discussion the third assumption of "perfect competition in the market", because it can be considered validated by the large number of investors participating in the market. Instead, focusing on the first two assumptions, we can clearly see the criticisms of the EMH: both the academic research and the empirical evidence prove that investors don't always take their decisions rationally but, most of the times, in a biased manner and those *biases* have been identified in behaviors like the representative bias[5], the overconfidence bias[6] and also the well-known risk-aversion behavior in contrast to the risk-neutrality assumption of the EMH[7][8][9]; moreover, in this new world governed by *big-data*, information is not shared democratically and freely among all market participants and, even if one argues that's the case, there would still be the problem of processing all that information as fast as those disposing of the right technology to do it. Aligning with that consideration, O'Hara[10] pointed out how portfolio managers need to think beyond the information acquisition problem because, today, there are technologies able to get fundamental information more quickly than any human analyst can:

3. E.F. Fama, "Efficient Capital Markets: A Review of Theory and Empirical Work", *The Journal of Finance*, vol. 25, no. 2, pp. 383-417, 1970.
4. P. A. Samuelson, "Proof That Properly Discounted Present Values of Assets Vibrate Randomly", *The Bell Journal of Economics and Management Science*, vol. 4, no. 2, pp. 369-374, 1973.
5. D. Kahneman, J.L. Knetsch, R.H. Thaler, "Anomalies: The Endowment Effect, Loss Aversion, and Status Quo Bias", *Journal of Economic Perspectives*, vol. 5, no. 1, pp. 193-206, 1991.
6. J. Scott, M. Stumpp, P. Xu, "Overconfidence Bias in International Stock Prices", 2003.
7. S. LeRoy, "Risk Aversion and the Martingale Property of Stock Prices", *International Economic Review*, vol. 14, no. 2, pp. 436-46, 1973.
8. M. Rubinstein, "The Valuation of Uncertain Income Streams and the Pricing of Options", *Bell Journal of Economics*, vol. 7, no. 2, pp. 407-425, 1976.
9. J. Robert, E. Lucas, "Asset Prices in an Exchange Economy", *Econometrica*, vol. 46, no. 6, pp. 1429-1445, 1978.
10. M. O'Hara, "High frequency market microstructure", *Journal of Financial Economics*, 2014.

> "Purchasing early access to the information and processing them faster than others have a discriminative effect on the market: the common information of the EMH leaves its place to huge amount of data on the hand of those with the most sophisticated technologies".

Moving from these premises to structure this work, we consider different approaches widely used to predict and exploit the patterns of a financial market that may diverge from its efficient status.

Stock prices' predictions are traditionally based on two different approaches: a *technical* and a *fundamental analysis*, depending on the type of information they rely on.

Technical analysis: Technical analysis refers to the study of past prices' movements – or any measure derived from them – to predict their future behavior. That type of analysis assumes that there are patterns in the prices that repeat themselves through time and being able to spot them means being able to predict future price movements. One of the most widely used family of models exploiting those technical patterns includes the Autoregressive (AR) model, the Autoregressive Moving Average (ARMA) model and the Autoregressive Integrated Moving Average (ARIMA) model for linear and stationary time series. Despite their simplicity and robustness of results, those models fail in modelling more complex nonlinear systems like the financial time series. Those frameworks, indeed, leave space for more sophisticated nonlinear statistical methods and, given the recent progress in the field, for deep learning methods.

Among the best-known neural networks' architectures, Recurrent Neural Networks (RNN), in particular Long Short-Term Memory (LSTM) networks[11], have been the state-of-art for modelling dependencies in time series till recently. In 2019, O. B. Sezer et al.[12] published a survey on general trends in financial deep learning models and confirmed the absolute dominance of RNN-based models for financial forecasting tasks (accounting for almost 51% of the overall deep learning models). W. Bao et al[13] used LSTM networks to predict one-step ahead prices of six stock indices using as input variables different trading data, technical indicators and macroeconomics variables; the results obtained confirm the superiority of the LSTM in terms of predictive accuracy with an average MAPE of 0.011 for the S&P 500 Index on a training set of 6 years. Similarly, T. Fisher et al.[14] deployed LSTM networks to predict directional movements of the S&P 500 stock constituents from 1992 until 2015, finding that LSTM networks outperform memory-free classification methods like ran-

11. S. Hochreiter; J. Schmidhuber, "Long Short-Term Memory", *Neural Computation*, vol. 9, no. 8, pp 1735-1780, 1997.
12. O. Berat Sezer, M. Ugur Gudelek, A. Murat Ozbayoglu, *Financial Time Series Forecasting with Deep Learning : A Systematic Literature Review: 2005-2019*, Elsevier Ltd, 2019.
13. W. Bao, J. Yue,Y. Rao, *A deep learning framework for financial time series using stacked autoencoders and long-short term memory*, Boris Podobnik, 2017.
14. T. Fischer, C. Kraussm, "Deep learning with long short-term memory networks for financial market predictions", *European Journal of Operational Research*, vol. 270, no. 2, pp. 654-669, 2018.

dom forest (RAF), deep neural net (DNN) and logistic regression classifier (LOG) with mean returns of 0.46 percent before and 0.26 percent after transaction costs and a standard deviation of 0.0209.

Despite their notable task-specific performances, recurrent methods show limitations in modelling longer-term and complex relations in sequence data: the *vanishing and exploding gradient* problem is emphasized by RNNs while the *overfitting* problem gets worsened by the step-by-step processing of the LSTMs. In 2018, the Google team of DeepMind introduced a novel neural network architecture called Transformer [15] that, instead of processing data in an ordered manner, like sequence-aligned models, processes an entire sequence of data altogether and uses self-attention mechanisms to learn complex dependences in the sequence. The Transformer network soon became the state-of-the-art for Natural Language Processing (NLP) tasks and established cutting-edge results also in the financial field. Q. Ding et al.[16] were probably the first ones to apply a Transformer model in financial time series prediction. They compared the performance of their own Transformer variant with convolutional neural network, LSTM network, attention-based LSTM and original encoder-only Transformer using daily Nasdaq data and 15-minute data from China A-shares; they showed that their variant of the Transformer reached an accuracy of prediction of 57.3 percent versus the 56.01 percent of the original Transformer and the 53.81 percent of the LSTM, while the other neural networks performed more poorly. Keeping on with innovative Transformer variants meant to specifically address financial tasks, H. Zhou et al.[17] introduced the *Informer*, a Transformer-based structure that deals with the problem of long sequences by adding a layer of sparse attention to the original network in order to reduce the model complexity and the memory usage, still maintaining comparable performances; similarly, E. Ramos-Pérez et al.[18] introduced the *Multi-Transformer* network that merges several multi-head attention mechanisms to improve the stability and the accurateness of the output.

Moving on till most recent days, the new deep learning frontier is the so-called *interpretable deep learning*: despite better performances of Seq2Seq models (like the Transformer network) in comparison to iterative sequence models (like the LSTM network), the interpretability of their results still remains challenging and the urgency to read inside those so-called *black-boxes* has just started to take off[19]. A nov-

15. A. Vaswani, N. Shazeer, N. Parmar et al., "Attention Is All You Need", 31st Conference on Neural Information Processing Systems, p. 15, 2017.
16. Q. Ding, S. Wu, H. Sun et al., "Hierarchical Multi-Scale Gaussian Transformer for Stock Movement Prediction", Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, 2020.
17. H. Zhou, S. Zhang, J. Peng et al., "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting", p. 8, 2021.
18. E. Ramos-Pérez, P.J. Alonso-González, J.J. Núñez-Velázquez, "Multi-Transformer: A New Neural Network-Based Architecture for Forecasting S&P Volatility", *Mathematics*, 2021, 2021.
19. F. Fan, J. Xiong, M. Li, G. Wang, "On Interpretability of Artificial Neural Networks: A Survey" 2021.

el architecture, the Temporal Fusion Transformer (TFT)[20] introduced by Google in 2021, paved the way towards the interpretable results' objective: while obtaining significant performance improvements over state-of-the-art benchmarks, it enables new forms of interpretability on three main components of the output, specifically the variable selection network, the persistent temporal patterns and the significancy of the occurred events.

Fundamental analysis: One major limitation of technical analysis is that it is incapable of unveiling the rules that govern the dynamics of the market beyond price data. Fundamental approaches, on the contrary, seek information from outside market-historic data such as geopolitical, financial environment and business principles.

There have been many attempts to mine news data to better predict market trends. Early in 2015, X. Ding et al.[21] proposed a deep learning method for event-driven stock market prediction, extracting events from textual news and using deep convolutional neural networks to model both short-term and long-term influences of events on stock price movements; they showed to outperform baseline methods by nearly 6 percent, recording an accuracy of prediction of 64.21 percent. Later in 2018, Z. Hu et al.[22] designed a Hybrid Attention Networks (HAN) to predict stock trends based on the sequence of recent related news; their model achieved the best accuracy of 0.478 compared to attention-based models, RNN and RF and the highest annualized return of 0.611 obtained by investing in the top 40 stocks accordingly to the model versus a market performance of 0.04.

Another major aspect in market news mining is the sentiment analysis of public news and social media, used to predict market trends. C. Ko et al.[23] combined the technical analysis with the fundamental one by including news articles and PTT forum discussions to understand market participants' sentiment; they found that the inclusion of news and forums' data could improve the accuracy of the predictions by an average of 0.23, as measured by the Root Mean Squared Error (RMSE) metrics. Behind those models trying to capture the sentiment implied by written words, there's a neural network architecture that has established itself as a cutting-edge framework for text-classification tasks: the Bidirectional Encoder Representations from Transformers (BERT)[24].

20. B. Lim, S.O. Arik, N. Loeff, T. Pfister, "Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting", *International Journal of Forecasting*, vol. 37, no. 4, pp. 1748-1764, 2021.
21. X. Ding, Y. Zhang, T. Liu, J. Duan, "Deep Learning for Event-Driven Stock Prediction", *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.
22. Z. Hu, W. Liu, J. Bian, X. Liu, T.-Y. Liu, "Listening to Chaotic Whispers: A Deep Learning Framework for News-oriented Stock Trend Prediction", WSDM 2018: The Eleventh ACM International Conference on Web Search and Data Mining, 2018.
23. C.-R. Ko1, H.-T. Chang, LSTM-based sentiment analysis for stock price forecast, Feng Xia, 2021.
24. J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, vol. 1, 2019.

3. FORECASTING MODELS

In this section we provide the theorical bases behind the models we're going to use in the methodology: we start from the simple statistical ARIMA model and then we move towards more sophisticated deep learning models.

3.1 ARIMA

The Auto-Regressive Integrated Moving Average (ARIMA) model is a well-known statistical approach used to model dynamical systems which show the properties of linearity and stationarity.

In order to make a time series x_t stationary, we compute its d^{th} difference, where the d term represents the number of nonseasonal differencing operations required to eliminate its trend component; then, we treat the resulting time series y_t as a weakly-stationary one and we model it using the combination of two components:

- its lagged time series values: Auto-Regressive (AR) component;
- the moving average of its lagged forecasting errors: Moving-Average (MA) component.

Once obtained those components, the ARIMA model is entirely identified by the tuple (p, d, q), where p and q define the orders of AR and MA, and d specifies the order of the differencing equations. p and q orders must be chosen empirically; one of the most used statistics to check for the rightness of the selected orders is the Ljung-Box statistics that check the closeness of the residuals obtained with the selected ARMA (p, q) model to a white noise process.

Once the ARIMA (p, d, q) is correctly specified in all its components, we can model the differenced time series y_t through the following stochastic process:

$$y_t = \phi_0 + \sum_{i=1}^{p} \phi_i \, y_{t-1} + \varepsilon_t - \sum_{i=1}^{q} \theta_i \, \varepsilon_{t-i}$$

where yt is the d^{th} differenced time series, ϕ_i is the autoregressive component of order $p \geq o$, θ_i is the moving average component of order $q \geq o$ and $\{\varepsilon_t\}$ is a white noise process with mean zero and standard deviation σ_ε.

3.2 Deep Learning Models

The employment of neural networks to solve nonlinear problems wasn't the clear solution till just 25 years ago when machine learning finally started its breakthrough. To depict the context at that time, we bring back the analysis conducted by Zhang et al.[25] in 1998 when reviewing the state-of-the-art in forecasting with Artificial Neu-

25. G. Zhang, B.E. Patuwo, M.Y. Hu, "Forecasting with artificial neural networks: The state of the art", *International Journal of Forecasting*, vol. vol. 14, no. issue 1, pp. 35-62, 1998.

ral Networks (ANNs): despite the attractive features of ANNs and their great potentials, academic concerns where mainly driven by two unsolved questions:

- *Given a specific forecasting problem, how do we systematically build an appropriate ANN that is best suited for the problem?*
- *What's the best algorithm to train ANNs?*

Answering the first question meant knowing exactly how to select the most appropriate architecture of the ANN, given a specific problem formulation and input settings. Said differently, it meant finding the optimal neural network structure (right number of nodes, hidden layers and output nodes; right interconnections among nodes; right activation functions etc.), that would have consistently outperformed all the others.

Answering the second question, instead, meant finding that training algorithm that would have consistently guaranteed the optimal solution for a general nonlinear optimization problem in a reasonable amount of time. At the time the question was formulated, the most popularly used training method was the stochastic gradient descent, with the backpropagation algorithm used to efficiently compute the gradient[26]. However, with backpropagation the learning process was still subject the vanishing and exploding gradient problem.

That second question got answered just a little later in time, thanks to the discovery of new *optimization* and *regularization* techniques that made the breakthrough of deep learning methods possible. In the following bullet list, we present the initial techniques used by the research community and the most relevant innovations that replaced them:

- *sigmoid activation* functions were predominant but they made the vanishing gradient problem worse because they saturated neural network nodes, interrupting the learning process; the introduction of *Rectified Linear Unit* (ReLU) functions improved the computational efficiency of the learning process and partially solved the vanishing gradient problem by avoiding the nodes' saturation in the positive region;
- *Gaussian weight initializations* were the standard but brought activation functions very close to 0 or 1, hence enhancing the vanishing gradient problem; *Xavier initializations* solved the problem by performing a weight normalization dependent on the size of the input;
- the *stochastic gradient descent* method was very slow in the convergence; the *Adam optimizer*[27] method speeded it up by making use of Momentum[28] and Adaptive Learning Rates to gradually reduce the velocity of learning and avoid overstepping local minima;

26. M. Nielsen, Neural Networks and Deep Learning, 2018.
27. J.B. Diederik, P. Kingma, "Adam: A Method for Stochastic Optimization", International Conference on Learning Representations, 2015.
28. Y. Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence o(1/k^2)", *Mathematics*, 1983.

- finally, *regularization techniques* were introduced to avoid overfitting problems: *L-norm regularizations*[29][30] helped keeping the weights small so that the network behavior wouldn't change too much by changing few random inputs; *dropout*[31] techniques, instead, worked on the structure of the network, rather than on the cost function, by randomly masking out some network's nodes to minimize the risk of overfitting.

Thanks to those new optimization and regularization techniques, the learning process of ANNs improved to the point that also the academic community – at the beginning more skeptical about machine learning methods – did recognize the consistency and accuracy of the obtained results. Then, the focus shifted to the first question: the discovery of the best ANNs *structures*.

Narrowing our interest to the time series forecasting task, we compare the performances of the most recent state-of-the-art forecasting models:

- Recurrent Neural Networks (RNN), which perform direct sequence processing and obtain best results through LSTM networks, whose learning process is optimized to keep memory of longer sequencies;
- Sequence-to-Sequence (Seq2Seq) architectures, which solve machine learning tasks where both inputs and outputs are sequences and obtain best results through Transformer neural networks employing *attention mechanism* to correctly weight the sequence's components.

Recurrent Neural Networks: Unlike traditional feedforward neural networks, Recurrent Neural Networks (RNNs) are a family of neural networks specifically designed to process sequential data. Indeed, their peculiarity is the presence of recurrent loops that make the information flow round in loops rather than only forward in the network.

Mathematically, we can represent a RNN as a dynamical system s_t with $\boldsymbol{\theta}$ shared parameters. At each time step t, the status of the system depends on its status at the previous time step $t\text{-}1$:

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}; \theta)$$

The current and past values of the system are defined by both the input vector x and the hidden state vector h. The value of the system s_t's hidden state h_t is obtained from the function g_t, which takes the whole past sequence x_t as input and produces the current hidden state h_t:

29. A.E. Hoerl, R.W. Kennard, "Ridge Regression: Biased Estimation for Nonorthogonal Problems", *Technometrics*, vol. 42, no. 1, pp. 80-86, 2000.
30. R. Tibshirani, "Regression Shrinkage and Selection via the Lasso", *Journal of the Royal Statistical Society*, vol. 58, no. 1, pp. 267-288, 1996.
31. G.E. Hinton, N. Srivastava, A. Krizhevskyet al., "Improving neural networks by preventing co-adaptation of feature detectors", 2012.

$$\boldsymbol{h}_t = g_t(\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \boldsymbol{x}_{t-2}, \ldots, \boldsymbol{x}_2, \boldsymbol{x}_1)$$
$$= f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \theta)$$

where g_t can be factorized into repeated applications of the same *transition function* f_t and *parameters* $\boldsymbol{\theta}=\{W,b\}$, which are the learnt weights and biases shared across all time steps. This factorization makes it possible to learn a single model f that operates at all time steps rather than a separate model g_t for each time step.

Based on that definition of hidden state vector h_t, a simple RNN can be described through the following system of two equations, taking as parameters the Kx1 vector of inputs x_t, the Nx1 vector oh hidden state h_t and the Lx1 vector of outputs y_t:

$$\begin{cases} \boldsymbol{h}_t = f(\boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1}) = f(\boldsymbol{u}_t) \\ \boldsymbol{y}_t = g\big(\boldsymbol{W}_{hy}\boldsymbol{h}_t\big) = g(\boldsymbol{v}_t) \end{cases}$$

where W_{hy} is the LxN matrix of weights connecting N hidden units to L outputs, W_{xh} is the NxK matrix of weights connecting K inputs to N hidden units and W_{hh} is the NxN matrix of weights connecting N hidden units from time *t-1* to time *t*.

The first equation of the system $f(u_t)$ is called *observation function* and describes the hidden layer activation function; the second equation of the system $g(v_t)$ is called *state equation* and describes the output layer activation function. The RNN neural network learns those two functions through the Back-Propagation-Through-Time (BPTT) algorithm which updates the weight matrices by minimizing the cost function *C*:

$$\boldsymbol{W} = \boldsymbol{W} - \gamma \frac{\partial C}{\partial \boldsymbol{W}}$$

where γ is the learning rate. The minimization of the cost function in RNNs is done as in feedforward neural networks, hence by minimizing the backpropagated error terms from the output layer to the first hidden layer. However, there's a problem arising with the presence of recurrent loops: since weight matrices are shared and kept equal across all time steps of recurrent loops, going backward in the network is equivalent to multiply the same weight *w* by itself many times (w^t), causing the product to either vanish or explode depending on the magnitude of *w*. In particular, the gradient of longer-term dependences has exponentially smaller magnitude than the gradient of shorter-term ones: this makes it more difficult and time expensive to learn longer term patterns whose signal is hidden by the smallest fluctuations of the shorter term interactions. In 1994, Y. Bengio et al.[32] showed that gradient-based optimization techniques become increasingly difficult when we increase the span of dependence, making the probability of a successful RNN training process rapidly reach 0 for sequences of length of only 10- or 20-time steps.

32. Y. Bengio, P. Simard, P. Frasconi, "Learning long-term dependencies with gradient descent is difficult", *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157-166, 1994.

Various approaches have been proposed over time to reduce the difficulty of learning longer term dependences through RNNs. In 1997, S. Hochreiter and J. Schmidhuber introduced a novel, efficient, gradient-based method called Long Short-Term Memory (LSTM)* that solved the vanishing and exploding gradient problem through the application of *gating structures* that control the information flow in the network.

LSTM: Long Short-Term Memory (LSTM) neural networks are a type of RNN employing *three gates* to modulate the information flow across memory cells. Each gate is composed of a sigmoid neural network layer and an element-wise multiplication; the sigmoid layer returns a number between 0 and 1 which determines how much information is passed through the gate itself. The three gates present in each LSTM cell are the following:

1. *Forget Gate*: it determines how much information from the previous hidden state h_{t-1} and the current input x_t must be forgotten;
2. *Input Gate*: it decides what of the new information coming from the previous hidden state h_{t-1} and the current input x_t is to be stored in the current *cell state segment* C_t, hence updating the previous C_{t-1}; the cell state segment of each LSTM unit is responsible for the *long-term memory* of the NN;
3. *Output Gate*: it controls the information to be stored into the current hidden state h_t, based on the previous hidden state h_{t-1}, the current input x_t and the modified cell state C_t; the hidden state h_t is passed to the next LSTM cell as the channel responsible for the *short-term memory* of the NN.

Keeping the two types of memory separated — the long-term one captured by the cell state C_t and the short-term one by the hidden state h_t — allows the learning process to be aware of the fluctuations of the two without incurring in the vanishing or exploding gradient problem.

The figure below represents graphically the structures just described.

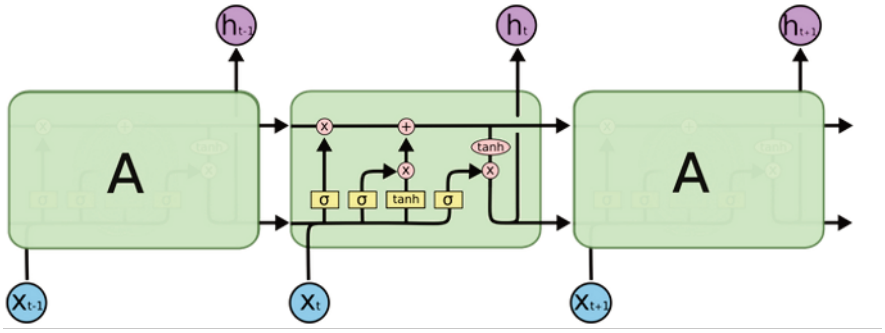


Figure 1: LSTM Architecture

* See note 11.

There's a major problem in the sequential nature of LSTMs that can't be solved through the gating mechanism: a cell state C_t can be computed only after having computed all cell states $\{C_{t-1}, C_{t-2}\ldots\}$ at previous time steps. This means that not only parallelization is not possible but also that LSTMs tend to experience overfitting problems: indeed, they propagate the error term of one-step-ahead prediction into accumulated errors of multiple-steps-ahead predictions.

In 2017, the Google Brain team proposed a new simple network architecture that dispenses with recurrence entirely, shows superior quality in the predictions and is more parallelizable: the Transformer neural network*.

Transformer: The Transformer neural network is a Seq2Seq architecture that associates an input sequence to an output sequence. It's composed of two main structures: an Encoder and a Decoder.

The *Encoder* is composed of an input layer, a positional encoding layer and a stack of N=6 identical encoder layers. The *input layer* maps the input sequence $(x_1,\ldots, x_n)$ to a vector of dimension $d_{model}=512$ called "embedded vector"; the *positional encoding* layer adds to the input vector the information about the position of each input into the sequence: a necessary step to keep memory of the relative positioning of inputs which are taken all together as a sequence rather than one by one like in RNNs; finally, the six identical *encoder layers* are all composed of two sub-layers:

- *a Multi-Head Attention* sub-layer:
  the capability of the network to capture nonlinearities in the input sequence lies mainly in the attention modules. Within each attention module, an entry of the sequence named "query" (Q) is compared to all other sequence entries named "keys" (K) through the dot-product attention, scaled by the d_k embedding dimensionality; the output of the dot-product is then used to weight the relevance of the sequence entries named "values" (V) with respect to the specific "query" (Q). Hence, the self-attention mechanism takes the following form:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

  The self-attention mechanism is also parallelizable: the scaled dot-product attention can be performed in parallel on h linear projections called "heads" of the queries, keys and values. The outputs are d_v-dimensional values, which are concatenated and projected to be passed to the next sub-layer. This is the so-called *multi-head attention* mechanism, which allows the model to simultaneously consider the difference representation subspaces of each input at different positions. It takes the following form:

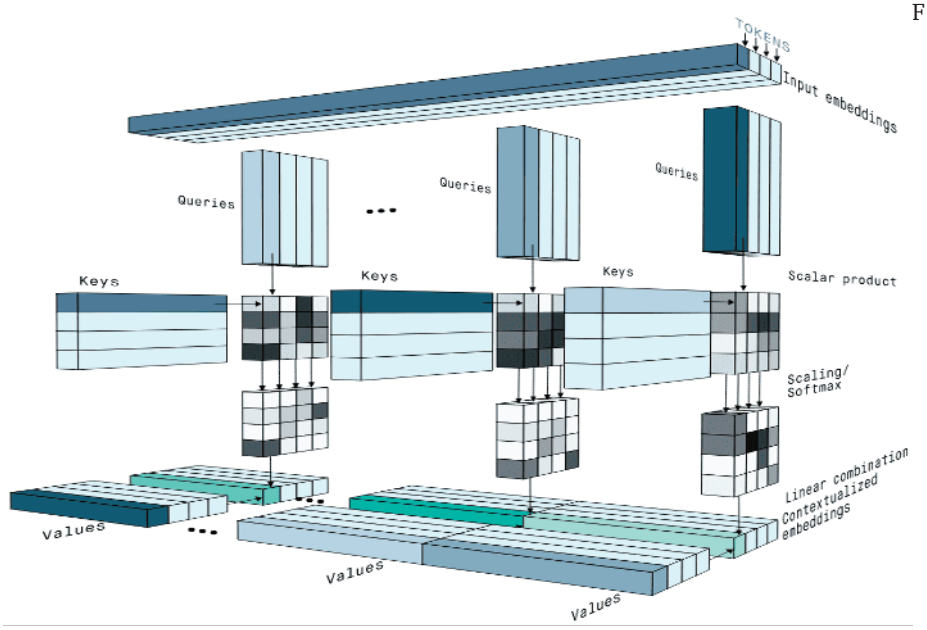
* See note 15.

$$MultiHead(Q,K,V) = Concat(head_1,\dots,head_h)W^O$$

$$where\ head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

where the projections are parameter matrices $W_i^Q \in R^{\text{dmodelxdk}}$, $W_i^K \in R^{\text{dmodelxdk}}$, $W_i^V \in R^{\text{dmodelxdv}}$ and $W_i^O \in R^{\text{hdvxdmodel}}$. The standard Transformer makes use of $h=8$ heads of attentions with dimensions $d_k=d_v=dmodel/h=64$.

Fi-



gure 2: Multi-Head Attention mechanism

- *a fully connected Feed-forward* sub-layer:
the feedforward layer is a fully connected Feed-forward network consisting of two linear transformations with a ReLU activation in between:

$$FFN(x) = \max(0, xW_1 + b_1)\,W_2 + b_2$$

where x is the input to the feedforward layer. This sub-layer allows to transform the attention vectors into a form that is acceptable to the following encoder or decoder layer.
On the other side, the *Decoder* is composed of an output embedding layer, a positional encoding layer, N=6 identical decoder layers and an output layer. The target sequence is first fed to the *output embedding* and the *positional encoding* layers, which produce an encoded representation for each element of the target sequence. Then, that embedded sequence is given as input to the six identical *decoder layers*, each of them composed of three sub-layers:

- *a Masked Multi-Head Attention* sub-layer:
  while the Encoder is designed to attend all inputs in the sequence, the Decoder is modified to attend only earlier positions in the sequence: a *mask* is applied to future positions so that the Multi-Head Attention cannot look at the element it has to predict or at subsequent elements in the target sequence.
- *a Multi-Head Attention* sub-layer:
  it works identically to the Encoder Multi-Head Attention layer but receives different sources of inputs: the queries from the previous decoder layer, and the keys and values from the output of the Encoder.
- *a fully connected Feed-forward* sub-layer.

Finally, the output layer transforms the output of the last decoder layer through a fully connected layer and a soft-max layer to generate the prediction of the next element of the output sequence.

From the above description of the Transformer neural network, we can immediately see numerous differences with RNNs and also the advantages that derive from using Seq2Seq models:

- RNNs takes the sequential data element by element (one step at time) and process them sequentially, starting from the beginning of the sequence to its end; Transformers take the entire input sequence all at once and preserve its sequential nature through positional embeddings, hence making themselves much more parallelizable than RNNs;
- RNNs can remember of forget parts of the previously stored information, depending on whether they find them important or not for the current memory state; Transformers rely on two specialized units (Encoder and Decoder) to perform the two tasks separately: being able to access all relevant past information at each time step as summarized by the Encoder and exploiting them to predict future values through the Decoder;
- RNNs, including LSTMs, decide which information of longer-term dependences is worth to remember in the current cell state but cannot assign more importance to some parts of the input sequences; Transformers, instead, use the Encoders to capture dependences on the input sequences and summarize the weights assigned to each of them to the Decoders.

However, there's a major drawdown in using Transformers to process longer time-series: since the representation vector computed by the Encoder and given as input to the Decoder is a summarized representation of all information captured by the Encoder, that representation becomes increasingly poor when longer-term dependences need to be encoded into the intermediate vector.

In 2019, the Google team of Deep Mind published a new architecture of the original Transformer called *Temporal Fusion Transformer* (TFT) with the specific objective of making it more adequate to encode longer term dependences, together with some other innovative objectives we describe in the next paragraph*.

* See note 20.

Temporal Fusion Transformer: The TFT neural network was specifically designed to solve two main characteristics of realistic scenarios encountered in multi-horizon forecasting tasks: the first is the *heterogeneity of inputs* to analyze; the second is the necessity to generalize the findings by looking into the mechanics that led the NN to formulate specific predictions, where the mechanics can either be the static patterns among input variables or their temporal dependences: in general, we could say we're interested in the *interpretability of NN results*.

The architecture of the TFT is very complicated as meant to reach state-of-art results for multi-horizon forecasting tasks by using the best components of LSTMs and Transformers but with the flexibility of Gated Residual Networks (GRN) that allow to skip all unnecessary components of the architecture whenever the task doesn't require that additional complexity.

Focusing only on the two main characteristics of the TFT described above, we start by explaining where its ability to deal with heterogeneous inputs comes from. The TFT is specifically designed to digest and exploit information about the category the fed input sequence belongs to:

- *observed past inputs*:
  time-dependent inputs that can only be observed in the past and be known as soon as they occur, but cannot be anticipated;
- *known future inputs*:
  time-dependent future inputs that are known also in the past, like specific future times, events or occurrences;
- *static covariates*:
  features that don't change over time and describe the context where time-dependent features develop, that are either geographical contexts, temporal ones or categorical types of context.

The figure below shows how those features are allocated in the timeline fed to the TFT.

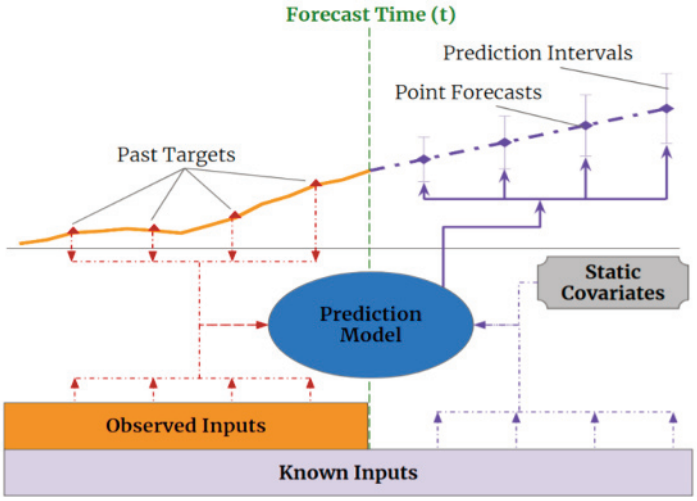


Figure 3: Temporal Fusion Transformer inputs

Finally, we describe the second characteristic of the TFT of making its results interpretable. To introduce the concept, we rely on the words used by the Google team in the original paper:

> "Several Deep Learning methods have been proposed, but they're typically "black-box" models which don't shed light on how they use the full range of inputs present in practical scenarios. In this paper, we introduce the TFT, a novel attention-based architecture which combines high-performance multi-horizon forecasting with interpretable insights into temporal dynamics".

The TFT allows to draw intuitions from three levels of interpretability, specifically to identify the following aspects of the obtained outputs:

- *globally important variables*:
  the innovation of the TFT come from the combination of a *Variable Selection Network* with an *Interpretable Multi Head Attention* layer. The Variable Selection Network allows to select the most relevant input variables at each time step and to suppress the noisy ones that could negatively impact the quality of the predictions; after that step, all selected variables belonging to each input category are fed into the Interpretable Multi Head Attention layer that performs a scaled-dot product attention with multiple heads as the original Transformer does but outputs each feature weight as the average of the weights assigned to that feature by all heads: that average allows to store each feature's weight at each time step, hence to understand which are the globally important variables for the TFT given the task.

- *persistent temporal patterns*:
  the persistency of temporal patterns in input sequences is measured by looking at the features contribution to future predictions at defined fixed lags in the past, given various future horizons. Whether there's consistency or variability in the relations between the attention weights assigned to input sequences determines the persistency of a specific temporal pattern or not.

- *significant events and regime changes*:
  the possibility of identifying regime changes in the input dependences is strictly correlated to the previous aspect of interpretability: sudden changes in persistent temporal patterns indicate the occurrence of significant events or changes in regimes that have an impact on subsequent predictions because relations among features' weights have changed.

Still not tested in many concrete applications, the Temporal Fusion Transformer promises to outperform its benchmarked forecasting models in predictions' accuracy while encountering one major desire of today research community: the interpretation of results.

4. PROBLEM STATEMENT

We formulate Ferrari stock price forecasting as a supervised machine learning task:
given an input time series X containing N daily data $\{X_{(t-N)}, \ldots, X_{(t-1)}, X_{(t)}\}$*, the forecasting model returns an output time series Y containing M steps-ahead predictions* $\{Y_{(t+1)}, \ldots, Y_{(t+M-1)}, Y_{(t+M)}\}$*, with* $M = 1$ *for Sequence models and* $M > 1$ *for Seq2Seq models.*

Each data point $X_{(i)}$ is a vector containing multiple features and each prediction $Y_{(i)}$ is a scalar representing the forecasted Ferrari closing price.

We employ four different forecasting methods which we compare on the Root Mean Square Error (RMSE) of their results:

$$RMSE = \sqrt{\sum_{i=1}^{M} (\hat{y}_i - y_i)^2}$$

The best performing method is the one that minimizes the RMSE of out-of-sample predictions. We select the ARIMA model as our task's baseline; we further select the LSTM network, the Transformer network and the Temporal Fusion Transformer network as deep learning models.

We expect deep learning models to outperform the baseline ARIMA model; we further expect Transformer-based models to obtain better performances than models without Seq2Seq structures and, among them, the Temporal Fusion Transformer to do better than the original Transformer as actually designed to predict long sequences with irregular temporal patterns.

5. DATA AND METHODOLOGY

5.1 Features Selection

We utilize daily data downloaded from Refinitiv Workplace* from the 23rd October 2015, immediately after Ferrari IPO at the NYSE, to the 1st February 2022, covering a total period of 2293 business days.

The *features selection process* directly involved the Investor Relation Team of Ferrari, whose business knowledge helped identifying three main sources of data to draw from:

1) *Company-specific data*:
   we combine a technical approach with a fundamental one, based on the evidence of greater alphas generated when using combined strategies rather than standalone strategie [33] [34]:

* Refinitiv, from Thomson Reuters, is one of the largest global providers of financial market data and infrastructures. Refinitiv Workspace is the Refinitiv software used in this work to download market data through the API Keys.

33. K. Li, P. Mohanram, "Fundamental Analysis: Combining the Search for Quality with the Search for Value", *Contemporary Accounting Research*, vol. 36, no. 3, pp. 1217-1927, 2018.

34. Z. Zhou, M. Gao, Q. Liu, H. Xiao, "Forecasting stock price movements with multiple data sources: Evidence from stock market in China", *Physica A: Statistical Mechanics and its Applications*, vol. 542, no. 15, 2020.

- in terms of *technical analysis*, we use Ferrari historical stock prices (OHLC) plus volume and derived technical indicators such as momentum, volatility and cycle-based indicators;
- in terms of *fundamental analysis*, we use Ferrari balance-sheet ratios such as profitability, liquidity, leverage and efficiency ratios and financial analysts' estimates on target-prices and target-EPSs to incorporate some information about the sentiment of the market.

2) *Sector-specific data*:
among the sectors Ferrari is involved into or may be affected by, we consider the *automotive*, *luxury* and *oil and metal* sectors; for our prediction task, we want to construct three proxies – one per sector – where the components of each proxy are those showing major explanatory power over Ferrari returns; to do it, we proceed as follows:
   1. we download the daily closing prices of the listed companies that are global leaders for the automotive and luxury sectors and the daily closing prices of the most traded futures on the oil and metal sectors;
   2. we construct a correlation matrix between the downloaded data and Ferrari daily closing prices, all transformed into log-returns;
   3. we remove from our dataset all the time series that show a correlation coefficient with Ferrari lower than 0.2[35], as we assume they won't add any relevant information for our prediction task.

The time-series not removed from the original dataset are taken as features.

3. *Global and Macroeconomic data*:
assuming the significance of global and macroeconomic variables' impact on Ferrari stock prices [36 37 38 39 40], we consider three different classes of global and macroeconomic variables: global equity indices, interest rates and exchange rates.
We download different financial instruments representing the three classes above and we conduct a linear correlation analysis of each of them with Ferrari stock price, as done with the sector-specific data. We keep as features only those exhibiting a correlation coefficient greater than 0.2.

35. G. Chandrashekar, F. Sahin, "A survey on feature selection methods", *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16-28, 2014.
36. J. Lintner, "Inflation and Security Returns", *The Journal of Finance*, vol. 30, no. 2, pp. 259-280, 1975.
37. J.F. Jaffe, G. Mandelker, "The 'Fisher Effect' for Risky Assets: An Empirical Investigation", *Journal of Finance*, vol. 31, no. 2, pp. 447-458, 1976.
38. E.F. Fama, G.W. Schwert, "Asset returns and inflation" *Journal of Financial Economics*, vol. 5, no. 2, pp. 115-146, 1977.
39. N.-F. Chen, R. Roll, S.A. Ross, "Economic Forces and the Stock Market", *The Journal of Business*, vol. 59, no. 3, pp. 383-403, 1986.
40. S.J. Grossman, R.J. Shiller, "The Determinants of the Variability of Stock Market Prices", *The American Economic Review*, vol. 71, no. 2, pp. 222-227, 1981.

From the features' selection process, we obtain a total of 56 features that will constitute the input vector for our forecasting models.

The selected features are listed below.

TABLE 1: LIST OF THE 56 FEATURES OF THE INPUT DATASET

Historical Prices and Technical Indicators	Fundamental Indicators	Automotive Sector	Luxury Sector	Oil and Metal Sectors	Global and Macro data
Open	EPS	Volkswagen	LVMH	Platinum	S&P 500
High	DY	Ford	Kering	Palladium	VIX
Low	PE ratio	Stellantis	Estee Lauder	Copper	NASDAQ 100
Close	PCF ratio	Renault	C. F. R.	WTI	EuroStoxx 50
Volume	ROS ratio	Daimler	PVH	Brent	FTSE MIB
SMA5, SMA10, SMA20	Mkt. Cap.	Bayerishe	Swatch		EUR/USD
EMA20	Target price	GM	Hermes		LIBOR
CCI	Target EPS	BMV	Tapestry		
RSI		Tesla	Burberry		
		Continental	Moncler		
			L'Oreal		
			Luxottica		
			Ralph Lauren		
			Capri Hold.		
			Tiffany		

All data are transformed into *log-returns* to benefit from a comparable metric and avoid biases when fed into the forecasting models.

5.2 Dataset Splitting and Labelling

To produce a labeled dataset out of the original features dataset, we need to create the triplets of:

1. *training set*: used to fit the forecasting model;
2. *validating set*: used to tune the model's hyper-parameters;
3. *testing set*: used to test the model on out-of-sample data.

Instead of using a fixed-ratio approach, we choose to employ a *fixed-length sliding window approach* to construct multiple triplets of training, validating and testing sets: keeping constant the size of the three sets to a length that is smaller than the total length of the original dataset, we move each window across the original dataset of M steps per time to create multiple triplets of sets (where M is the number of steps-ahead predictions required to the forecasting model).

This approach allows to "artificially" increase the length of the original dataset and that's very helpful when we deal with features that don't cover a period of daily observations long-enough to ensure good generalizing properties of the forecasting model: in our case, few more observations than 6 years may have led to an algorithm forced to learn from a too short training set with too many features and, consequently, to overfitting problems.

The chosen proportions of the three sets in each triplet are the following:

*lenght(sliding window)= length(original dataset) *25%*
*lenght(training set)= length(sliding window) *80%*
*lenght(validating set)= length(sliding window) *10%*
lenght(testing set)= length(validating set)

so that, each training set accounts for a little more than 1 business year (exactly 315 days) and each validating and testing dataset for approximately 2 business months (exactly 40 days).

The figure below gives an intuitive idea of the obtained training, validating and testing triplets:

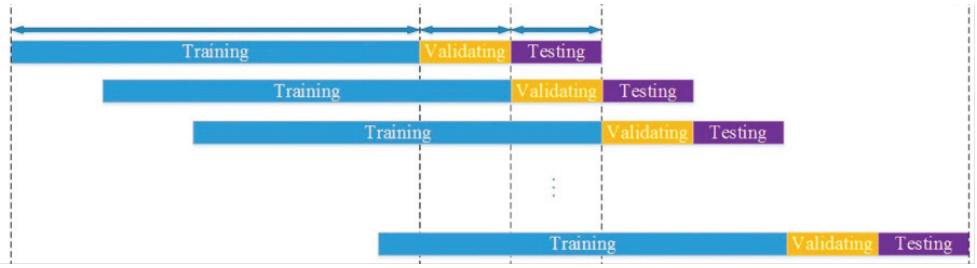


Figure 4: Sliding-window

Running the sliding window on the training set returns samples with features and labels, which are the previous N and next M observations respectively. Validating and test samples are also constructed in the same manner for model evaluation.

5.3 Pre-Processing

When dealing with real-world data, the chances of them being poor in quality or somewhere missing are high and can lead to misleading results. The objective of *pre-processing techniques* is exactly to improve the overall quality of the data so that both the robustness of the model and the training efficiency of the learning algorithm result increased[41].

In this work, the main stages of the data preprocessing are the *data standardization*, the *Wavelet transform* and the *Stacked Autoencoder* neural network.

Data Standardization: The training, validating and testing datasets are all scaled using the *Robust standardization* technique which rescales all the features in the datasets so that their distributions have mean 0 and standard deviation 1:

$$X_{standardized} = \frac{X - \mu}{\sigma}$$

where μ is the mean of the feature's distribution and σ its standard deviation.

41. S. Kotsiantis, D. Kanellopoulos, P.E. Pintelas, "Data Preprocessing for Supervised Leaning", *International Journal of Computer and Information Engineering*, vol. 1, no. 12, 2007.

Differently from the classic standardization, the Robust standardization technique also accomplishes the need to account for the presence of outliers by computing the mean and the standard deviation of each feature on the values belonging to the interquartile range of its distribution (from the 25^{th} to the 75^{th} quantiles), hence completely ignoring the outliers.

Since the algorithm is not allowed to look at the values of the validating and testing datasets as they have to remain "unseen" for learning purposes, the mean and standard deviation of each feature's distribution are computed on its training set and then used to transform not only the training dataset but also the respective validating and testing dataset.

Wavelet Transform: Financial time series are typically noisy and nonstationary. Those two characteristics together can make the useful signal of the time series even noisier than the underlying noise and, when it happens, traditional denoising techniques prove to be ineffective. James B. Ramsey clarified the problem and identified a possible solution in his remarkable paper "*The contribution of wavelets to the analysis of economic and financial data*", published in 1999[42]:

> "Traditionally in economic analysis the assumption has universally been made that the signal f(t) is smooth and the innovations $\varepsilon(t)$ are irregular. Consequently, it's a natural first step to consider extracting the signal f(t) from the observed signal $j(t) = f(t) + \varepsilon(t)$ by locally smoothing y(t).
> However, when the signal is as, or even more, irregular than the noise, such a procedure no longer provides a useful approximation to the signal. The process of smoothing to remove the contamination of noise distorts the appearance of the signal itself. However, when the noise is below a threshold and the signal is well above a threshold, one can isolate the signal from the noise component by selectively shrinking the wavelet coefficient estimates."

Indeed, the idea behind the wavelet denoising is to look at the time series as a signal to decompose into its low frequency (*approximation coefficients*) and high frequency (*detail coefficients*) components; once those components are identified, a threshold function and a threshold value are selected and applied to the decomposed time series: this filtering technique allows to keep the signal above the threshold, either shrinkage or set to zero the signal below the threshold and use the resulting signal to reconstruct the original time series as a denoised one.

This technique outperforms the well-known *Fourier transform* for the frequency analysis of financial time series. Indeed, the Fourier transform identifies all frequencies of a signal by decomposing it into a series of sines and cosines frequencies, but it doesn't provide any information about the time to which that frequencies correspond. Con-

42. J. B. Ramsey, "The Contribution of Wavelets to the Analysis of Economic and Financial Data", *The Royal Society*, vol. 357, no. 1760, pp. 2593-2606, 1999.

trarily, the *Wavelet transform* uses functions (*wavelets*) that are localized both in time and frequency, hence it overcomes the problem of dealing with sequential series.

The 3 factors necessary to proceed with the Wavelet transform are the following:

1) selection of a *wavelet basis*:
   the Wavelet transform represents any function as a superposition of a set of wavelets, which are small "waves" located in different times whose specific forms are determined by the so-called wavelet basis; those wavelets can be stretched and shifted to capture signals that are local in time and frequencies.
   More specifically, given a *wavelet mother* ψ, its *daughter wavelets* are obtained through the scaling parameter s and the translation parameter τ as:

$$\psi_{s,\tau}(t) = \frac{1}{\sqrt{2^s}}\psi\left(\frac{t - \tau * 2^s}{2^s}\right)$$

   Choosing the most appropriate basis function for the mother wavelet – given the characteristics of the signal under consideration – is the first step of the denoising task.

The figure below shows the shapes of the most common wavelet basis.

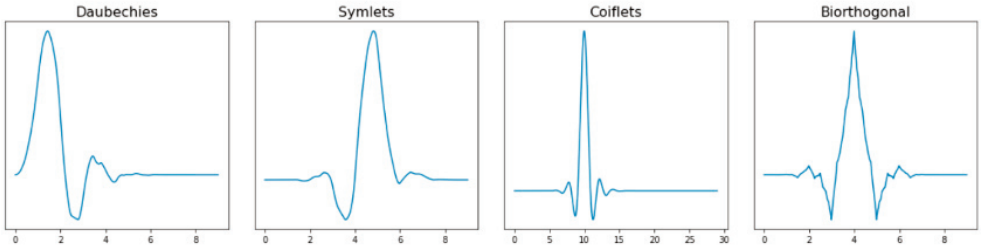


Figure 5: Discrete wavelet bases

2) determination of order and level of the *decomposition layers*:
   the next step after the wavelet basis's selection is the multilevel decomposition of the signal into its low frequency component and high frequency components, which results in one approximation coefficient and j detail coefficients, with j being the chosen decomposition level.

The figure below shows what happens at each level of decomposition.

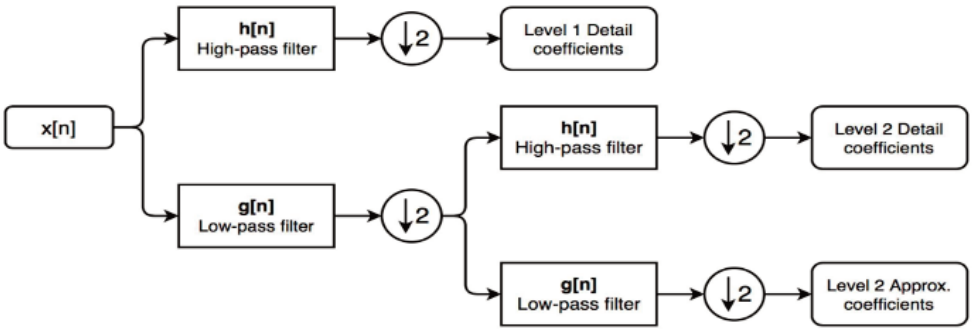


Figure 6: Wavelet decomposition at level 2

3) determination of the *threshold value* and *threshold function*
the thresholding step consists of applying a threshold value to the previously obtained detail coefficients with the scope of keeping only the useful signal and discard the irrelevant noise. The way the irrelevant noise is discarded depends on the selected threshold function: the *hard threshold function* sets to zero all the detailed coefficients smaller than the threshold value while the *soft threshold function* shrinks them towards zero. Both the functions show some problems: the hard thresholding makes the reconstructed signal have no continuity while the soft thresholding makes it too smooth. Hong-Ye Gao[43] proposed a different method, the *garrote threshold function*, which is asymptotically equivalent to the other two functions but shows smaller MSE and less sensitivity to small perturbations in the data.

Once the threshold value and the threshold function are selected, the signal can be reconstructed by performing the inverse of the Wavelet transform used in step 1). For this work, we test 4 different wavelet bases to denoise each feature of the dataset:
- Haar Wavelet;
- Daubenchies wavelet of order 3;
- Symlet wavelet of order 4;
- Coiflet wavelet of orders 3 and 4.

Given a specific wavelet basis, we decompose each feature through its maximum useful level of decomposition to obtain the detailed coefficients. To quantify that maximum level, we rely on the optimization function "*max_level*" of the Python built-in library PyWavelet.

In order to calculate the approximation and detail coefficients for both the validation and test sets, we want to avoid taking future information about the signal

43. H.-Y. Gao, "Wavelet Shrinkage Denoising Using the Non-Negative Garrote", *Journal of Computational and Graphical Statistics*, vol. 7, no. 4, pp. 469-488, 1998.

into account; to do it, we add one data point at a time to the training set and we calculate the coefficients of the obtained signal; then, we store the coefficients corresponding to the data point added. We perform this procedure until all data points in the validation and test sets have their respective coefficients calculated.

An example of the multi-level decomposition obtained when applying the Sym5 Wavelet transform to the standardized log-returns of Ferrari close prices is given by the following picture.

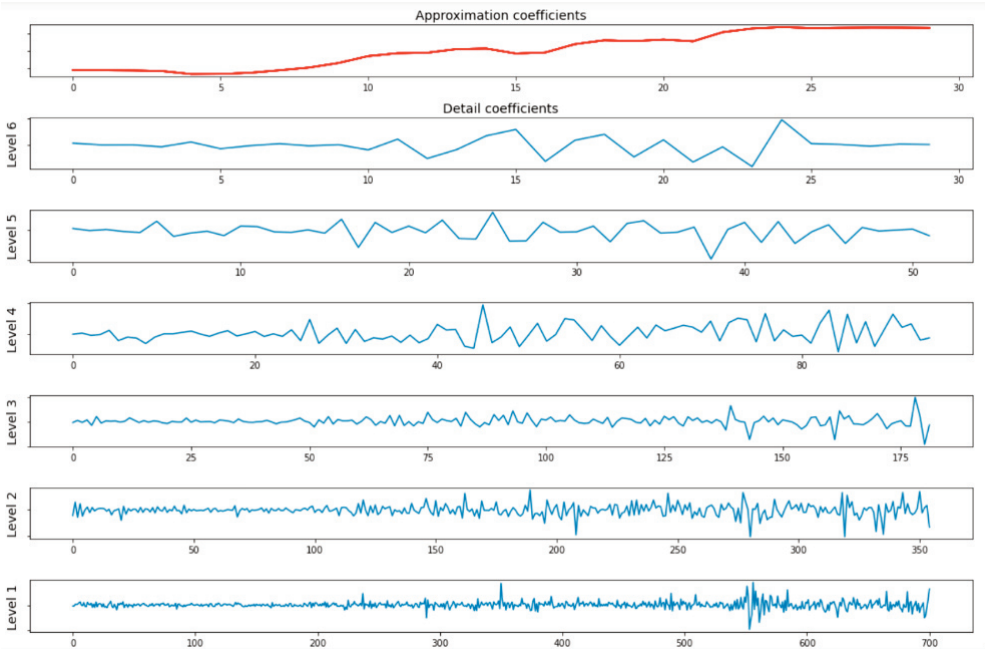


Figure 7: Sym5 6-levels wavelet decomposition

Once obtained the multi-level decomposition of each feature, we set the threshold value to one standard deviation of the feature's signal distribution and we select the garrote as threshold function. We apply the thresholding to all detail coefficients at each decomposition level and we reconstruct the signal through the inverse of the wavelet transform used for the decomposition.

Repeating the same procedure for all wavelet bases, we obtain a reconstructed signal for each feature's time series and for each tested wavelet basis. The way we select the best performing wavelet basis for our features dataset is by looking at its associated Signal to Noise Ratio (STN), where by "associated" we mean the average SNRs of all the features' time series reconstructed through that wavelet basis. Indeed, the SNR indicates how much of the useful signal is kept with respect to the removed noise: the higher the SNR, the better the denoising effect. Its mathematical formulation is the following:

$$SNR = 10\log\left(\frac{\sum_{j=1}^{M} x_j^2}{\sum_{j=1}^{N}(x_j - \hat{x}_j)^2}\right)$$

When applied to our features dataset, it returns the following SNR values for each tested wavelet basis:

Not denoised dataset	Haar	Db3	Sym5	Coif3	Coif4
0.2124	0.1994	0.2361	0.2357	0.2752	0.2538

where a SNR over 20 dB indicates that the signal has 100 times the power of the interfering noise. Based on the results above, we select the Coif3 as wavelet basis for out features dataset and we proceed with the denoising of the already standardized features. The reconstructed times series constitute the dataset used by the next preprocessing method.

Stacked Autoencoder: The process of determining the intrinsic dimensionality of a features dataset is called *Dimensionality Estimation (DE)* process and it may prove to be a fundamental step in the preprocessing phase to understand whether the dataset fed to the forecasting model contains all and only the relevant information, leaving out all the unnecessary data. Whenever the n-dimensional dataset turns out to have an intrinsic dimensionality of $p<n$, it means that the original dataset can be reduced in its dimensionality still preserving all the information of the higher dimensional dataset. In that case we proceed with a *Dimensionality Reduction (DR)* process.

DR can be particularly important when the input dataset contains multiple features that are highly correlated between themselves: feeding all of them to the forecasting model may lead to multicollinearity problems or may affect the robustness and the generalizing properties of the machine learning algorithm.

Multiples methods have been proposed for the DR process. The *Principal Component Analysis (PCA)* is among the most used linear methods: it uses the linear correlations among the features to find their direction of maximum variance and project the original dataset into a new space of fewer dimensions. However, since real world data don't lay on a linear manifold but rather on a non-linear one, nonlinear models may improve the accuracy of the dimensionality reduction process. Among the nonlinear models specifically thought for this task there's the Autoencoder neural network.

An *Auto-Encoder (AE)* is a neural network that learns an approximation to the identity function, so that the output $\hat{X}$ is as close as possible to the n-dimensional input X.

The AE's approximation works through an encoder and decoder structure: the encoder's hidden layers add increased levels of compression to the input vector; the inner hidden layer, when reached, returns the so-called *latent representation* of the input vector and passes the information to the decoder; finally, the decoder reconstructs the original input vector by de-compressing its latent representation through the same hidden layers used by the encoder.

The figure below gives a graphic representation of the structure just described.

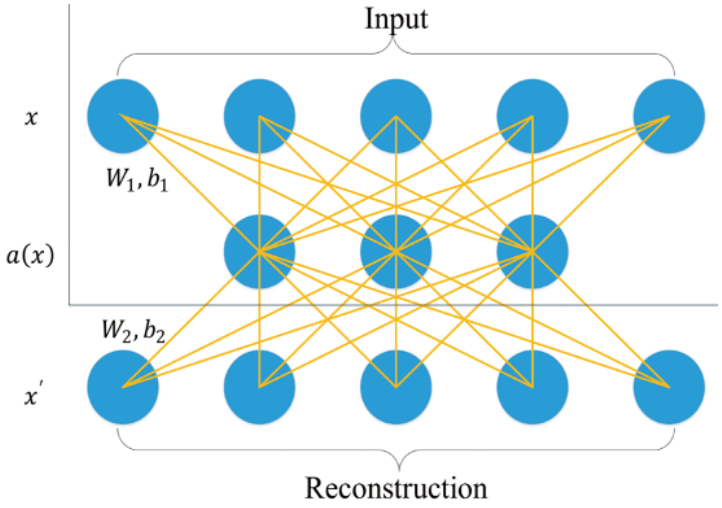


Figure 8: Single hidden-layer autoencoder

The AE learns by minimizing the difference between the original input vector and the reconstructed output: when that difference is minimized, the approximation of the original input is given by its latent representation. The features encoded by the latent representation are also called *deep features*.

To show how this approximation problem looks like using the parameters of the neural network, we provide the mathematical formulation of the simplest case of a single hidden-layer AE (like in the figure above). In that case, each input vector $x_i \in R^{nx1}$ is transformed into a hidden vector $y_i \in R^{kx1}$ for $k<n$ through the mapping:

$$\boldsymbol{y}_i = \sigma(W_1\boldsymbol{x}_i + \boldsymbol{b}_1)$$

where σ is the activation function (like a sigmoid function or a ReLU), W_1 is the first hidden layer's weight matrix and b_1 is the first hidden layer's biases vector.

The obtained hidden vector y_i is further transformed to the approximated input $\hat{x}_i \in R^{nx1}$ through another mapping:

$$\widehat{\boldsymbol{x}}_i = \sigma(W_2\boldsymbol{y}_i + \boldsymbol{b}_2) = \sigma(W_2\sigma(W_1\boldsymbol{x}_i + \boldsymbol{b}_1) + \boldsymbol{b}_2)$$

where W_2 and b_1 are the second hidden layer's weight matrix and biases vector, respectively.

Finally, the parameters $\{W_1, b_1, W_2, b_2\}$ are found by minimizing a cost function C that quantifies the difference between the original input x_i and the approximated $\hat{x}_i$:

$$min_{W,\boldsymbol{b}}\, C(W, \boldsymbol{b}; \boldsymbol{x}) = \; min_{W,\boldsymbol{b}}\, \frac{1}{2}\,|\boldsymbol{x} - \widehat{\boldsymbol{x}}|^2$$

where $W=\{W_1,W_2\}$ and $b=\{b_1,b_2\}$.

The higher the number of hidden layers used, the more complex the approximation. The need of performing a multi-layer dimensionality reduction may come from the fact of dealing with complex non-linear relationships among the input features so that using only one Auto-Encoder isn't enough to find their latent representation. The multi-layers autoencoders are called *Stacked Autoencoders (SA)* because they're obtained by training single autoencoders, then keeping only their respective hidden layers and stacking them one on top of the other to obtain the final multi-layer autoencoder.

In this work, we use a Stacked Autoencoder with 5 layers to perform the dimensionality reduction of our 56 features dataset, as suggested by Chen et al.[44]. We run the algorithm using the following parameters:

- Batchsize: 128
- Learning rate: 0.001
- Regularization: Gradient Clipping Technique
- Optimization: Adam optimizer [27]

The search-grid technique, applied to the number of hidden nodes per hidden layer, returns the best performance in terms of RMSE minimization in the following setup: input layer with 56 nodes by construction, first hidden layer with 25 nodes, second hidden layer with 35 nodes, third hidden layer with 25 nodes and output layer with 25 nodes. By reducing the dimensionality of the features dataset from 56 features to 25 deep features, we obtain a RMSE = 1.569 which we consider an acceptable loss given the required approximation.

The forecasting models used in the next section make use of the features dataset obtained from the three steps of the preprocessing phase: the data standardization, the data denoising through Wavelet Transform and the data dimensionality reduction through Stacked Autoencoder.

5.4 Deep Learning Framework

In this section we present the structure and the hyper-parameters of the three deep learning models being used; they're the result of the *fine-tuning process* carried out with the validation datasets.

LSTM: Input data are reshaped in the form [Batch size, Time steps, Features dimension], where the *batch size* is set equal to 64, the *time steps* to 24 days backwards for predictions of 1 to 5 daily steps ahead and the *features dimensions* to 25, which is the number of deep features obtained from the stacked autoencoder's dimensionality reduction. The LSTM neural network is structured as 4 stacked LSTMs with 1 hid-

44. Y. Chen, Z. Lin; X. Zhao et al., "Deep Learning-Based Classification of Hyperspectral Data", *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 6, pp. 2094-2107, 2014.

den layer each and 20 hidden neurons per layer. For the training process, we use the Adam optimizer with $\beta_1=0.9$, $\beta_2=0.98$ and $\varepsilon=10^{-9}$. The learning rate is chosen through the search-grid method and is set equal to 0.01. With respect to regularization techniques, we use a dropout rate of 0.2 for each sub-layer of the LSTM.

Transformer: We use the original structure of the Transformer composed of an Encoder and a Decoder, each of them with 6 identical stacked encoder and decoder layers, respectively. Each encoder layer is composed of two sub-layers, the multi-head attention sub-layer with a number of heads equal to $h=8$ and a position-wise fully connected feed-forward sub-layer. The dimension of the model is set equal to $d_{model}=512$ and all sub-layers produce outputs of dimension d_{model} to facilitate the residual connections. The dimensions of each head are chosen in such a way that $d_k=d_v=d_{model}/h=64$. The decoder layer is composed of the same two sub-layers of the encoder layer plus an additional sub-layer, the masked multi head attention sublayer, which applied subsequent masks to prevent the Transformer to look into the future when making the prediction.

For the training of the just described Transformer architecture, we use a batch size of 128 and the Adam optimizer with $\beta_1=0.9$, $\beta_2=0.98$ and $\varepsilon=10^{-9}$. The learning rate is chosen through the search-grid method and is set equal to 0.01. With respect to regularization techniques, we use a dropout rate of 0.2.

Temporal Fusion Transformer: Given the TFT capacity of dealing with more complex models, the input data fed to the TFT aren't reduced in their dimensionality: the dimension of the features dataset is kept equal to 56 rather than 25 like in the LSTM and in the original Transformer. This choice is driven by the objective of exploiting the interpretability results of the TFT, which means looking at the weights assigned to each identifiable variable rather than at those assigned to deep features of which we cannot understand the meaning from a posterior perspective.

Moreover, given the TFT capacity of approximating longer-term dependences among input data, we expand the lookback period to one business year, or 252 days of observations, and we keep the forecasting horizon of 1 to 5 daily steps ahead.

For the training of the TFT architecture, we use a batch size of 256 and the Adam optimizer with $\beta_1=0.9$, $\beta_2=0.98$ and $\varepsilon=10^{-9}$. The learning rate is chosen through the search-grid method and is set equal to 0.001. With respect to regularization techniques, we use a dropout rate of 0.4.

6. RESULTS

A univariate ARIMA model is used as baseline for our forecasting task, assuming that each daily Ferrari closing price is dependent on the previous p daily closing prices observations and q daily estimation errors. We select the order of the ARIMA model using the Akaike Information Criteria (AIC) to balance the goodness of fit with a certain simplicity and parsimony of the model. We choose the ARIMA (0,1,1), which is the standard Exponential Smoothing model. The model is trained with the first

two third of the dataset. The fitted parameters are then used on the full time series to make 1 to 5 daily steps ahead predictions.

We compare the ARIMA's performance with those of the LSTM, the Transformer and the TFT. Next table summarizes the RMSEs for each method, as well as the relative performance gain with respect to the one step-ahead forecast of the ARIMA model.

TABLE 2: SUMMARY OF MODEL PERFORMANCES AND RELATIVE CHANGES W.R.T. BASELINE MODEL

Model	RMSE 1 day forec.	RMSE 2 days forec.	RMSE 3 days forec.	RMSE 4 days forec.	RMSE 5 days forec.
ARIMA(0,1,1)	**0.521 (-0 %)**	0.672 (+28.98 %)	0.891 (+71.02 %)	0.877 (+68.83 %)	0.845 (+62.19 %)
Long Short-Term Memory (LSTM)	**0.642 (+23.22 %)**	0.651 (+24.95 %)	0.767 (+47.22 %)	0.893 (+71.40 %)	0.831 (+59.50 %)
Transformer	0.213 (-59.12 %)	**0.191 (-63.34 %)**	0.197 (-62.19 %)	0.228 (- 56.24 %)	0.312 (-40.12 %)
Temporal Fusion Transformer (TFT)	0.142 (-72.74 %)	0.144 (-72.36 %)	0.143 (-72.55 %)	**0.127*** **(-75.62 %)**	0.139 (-73.32 %)

The comparison suggests that multivariate deep learning models overall outperform the univariate ARIMA model. Within the three deep learning models, the Transformer-based models outperform the LSTM and the difference in performances becomes more evident as we increase the time steps of prediction. This suggests that the LSTM tends to experience overfitting problems in the shorter-term predictions and carries on those biases in the longer-term ones, where we see a poorer performance of the LSTM also in comparison to 5 steps-ahead predictions of the ARIMA model. This analysis indicates that attention mechanisms of Seq2Seq models highly increase the forecasting performance and guarantee generalizable properties of the forecasting model. Proceeding with our relative evaluation of results, we can see that the TFT shows better forecasting performance compared to the original Transformer, suggesting that the TFT can better capture complex dynamical patterns in the data even when the number of features and the number of backward time steps to look at are remarkably higher than benchmark models. The TFT is also the only deep learning model than consistently increases its performances with increased forecasting horizons, registering the best result in terms of RMSE for 4 steps ahead predictions: 0.0127 RMSE of 4 steps ahead predictions versus 0.141 RMSE of 1 step ahead predictions; moreover, the TFT's 0.0127 RMSE is to be compared against the same 4 steps ahead predictions' RMSEs of the Transformer (0.228), the LSTM (0.893) and the ARIMA(0,1,1) (0.877). Overall, the TFT has outperformed all other models in terms of RMSE of the predictions.

Finally, we focus on the best performing model, the TFT, to look at the *interpretability property* of its results. In the first figure below, we see the average attention patterns over the lookback period of 252 days used by the neural network to make predictions on 1 to 5 steps ahead forecasting horizons. We can see relevant spikes on 200 backward steps' observations and on the 150 ones. In general, we observe that higher spikes have higher attention patterns for longer term forecasting horizons rather than for the shorter ones, as if the effect of a change in underlying temporal dependences would take some time to completely reflect in Ferrari closing prices.

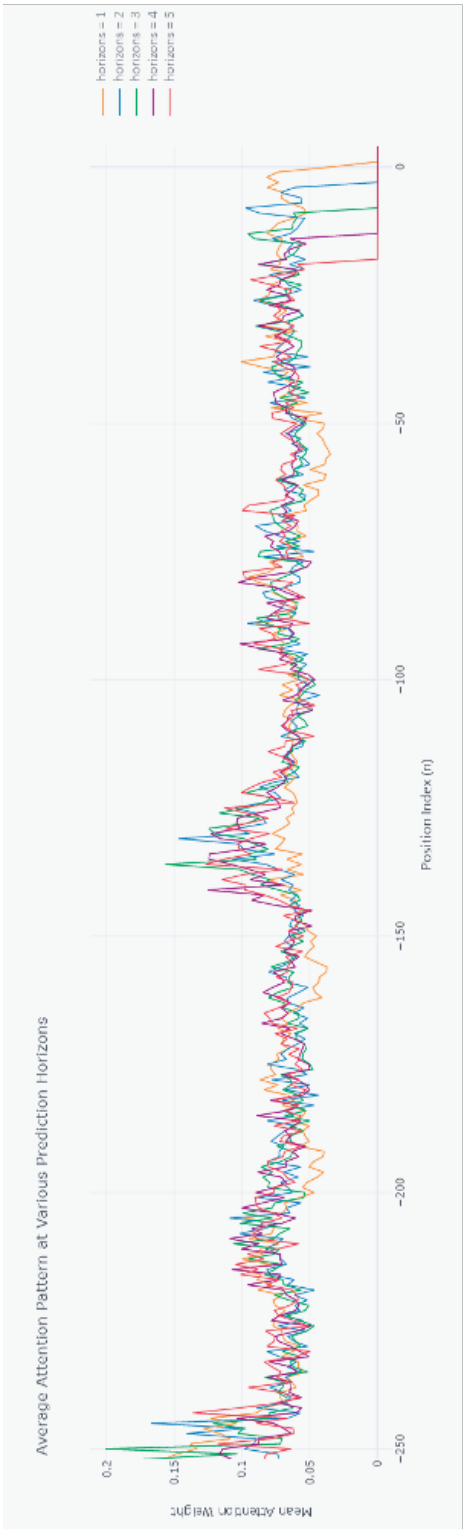


Figure 9: TFT Average Attention Patterns at various Prediction Horizons

In the second figure below, we present the reconstructed bar plot showing the average weights that the TFT assigns to the top three features of each input category for 4 steps-ahead predictions. In the Appendix at the end of this work, the complete reconstruction of the attention weights for all features of the input dataset is provided, completed with the detailed weights for each forecasting quantile (10%, 50% and 90%) and with the average of them all. We highlight some of the main aspects that come to notice when looking at the summary of the figure below:

- among the company-specific indicators, the technical ones show better ability to anticipate next movements of Ferrari closing prices, at least for the forecasting horizons considered in this work; we expect higher importance given to the fundamental ones for longer term predictions;
- overall, higher attention is given to the luxury sector rather than to the automotive one, as also confirmed by the table in Appendix. That aspect meets our expectations: the business of Ferrari addresses only the luxury segment of the market and there's no listed company belonging to the automotive sector that shares that same characteristic when considered a company as a whole; moreover, as declared by the Investor Relation team, Ferrari business is increasingly moving towards the fashion one and our analysis confirms the strategic importance of that decision as the drivers behind the fashion market already have an impact on Ferrari business;
- finally, the three global indices approximating the market trends show the least contribution to Ferrari prices' movements, when compared to all other top three features per category; again, this is not surprisingly since the target customer for Ferrari is presumably less affected by eventual downturn of the general market.

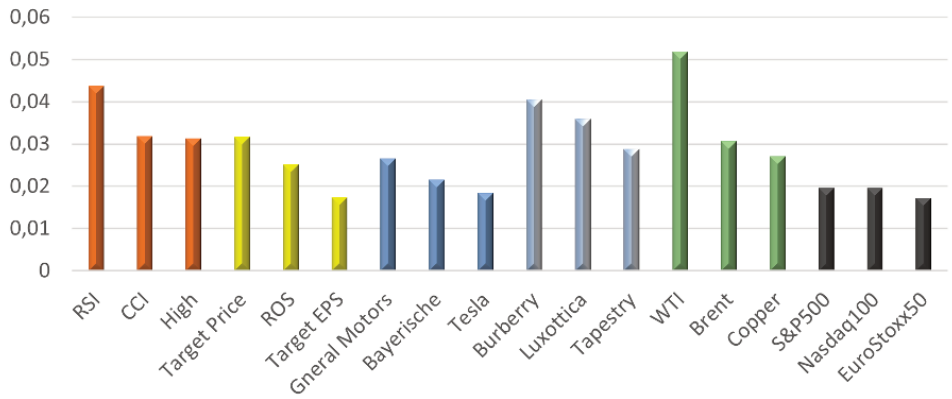


Figure 10: TFT Average Attention Weights at 4 steps-ahead predictions

7. CONCLUSION AND FUTURE WORK

In this work we presented *three deep learning approaches* to forecast financial time series data. In particular, we predicted Ferrari closing prices for 1 to 5 daily steps ahead horizons. We performed a univariate time series analysis through the ARIMA(0,1,1) model and we used that as our baseline to quantify the relative improvements of the multivariate analysis of the three deep learning models.

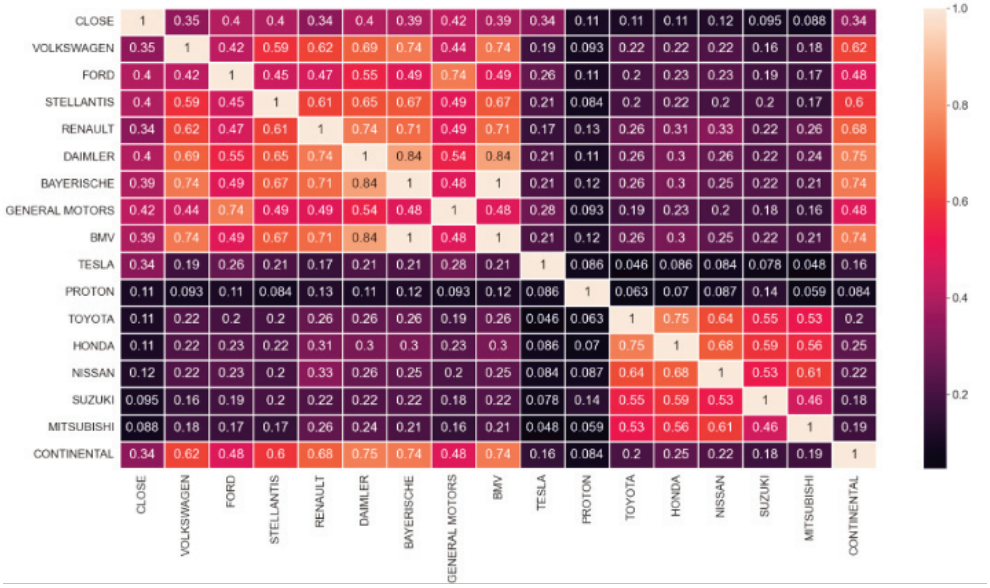
The core of the methodology has been the *data preprocessing* phase, done using the Robust Standardization to account for outliers, the Wavelet Transform to remove the unnecessary noise from the features' time series and the Stacked Autoencoder to reduce to dimensionality of the features dataset to its intrinsic dimensionality. Overall, it helped in creating a system that was robust enough to model the highly irregular and noisy nature of the financial time series; moreover, it resulted in a less prone to overfitting system with enhanced generalization capabilities.

The best performances were those of the two attention-based models, the *Transformer* and the *Temporal Fusion Transformer*, in comparison to the sequence-aligned LSTM method. Moreover, the TFT showed an improvement of almost 76% of the forecasts' RMSE compared to the baseline, hence confirming its superiority in learning complex dependencies of also longer-term time series.

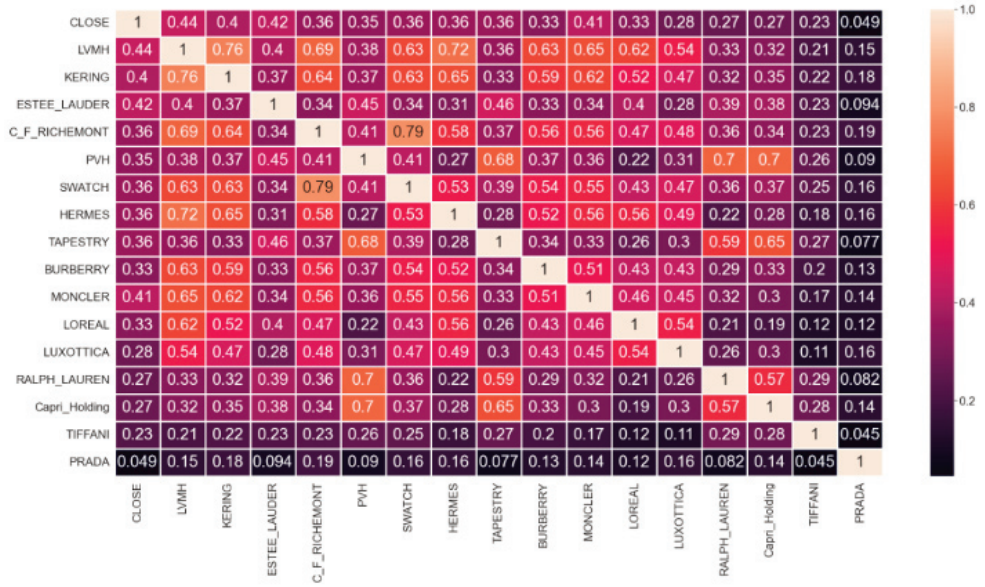
To conclude, we hypothesize that our approach can be further extended to model the sentiment of the market to better capture the direction of stock prices with attention-based deep learning models.

APPENDIX

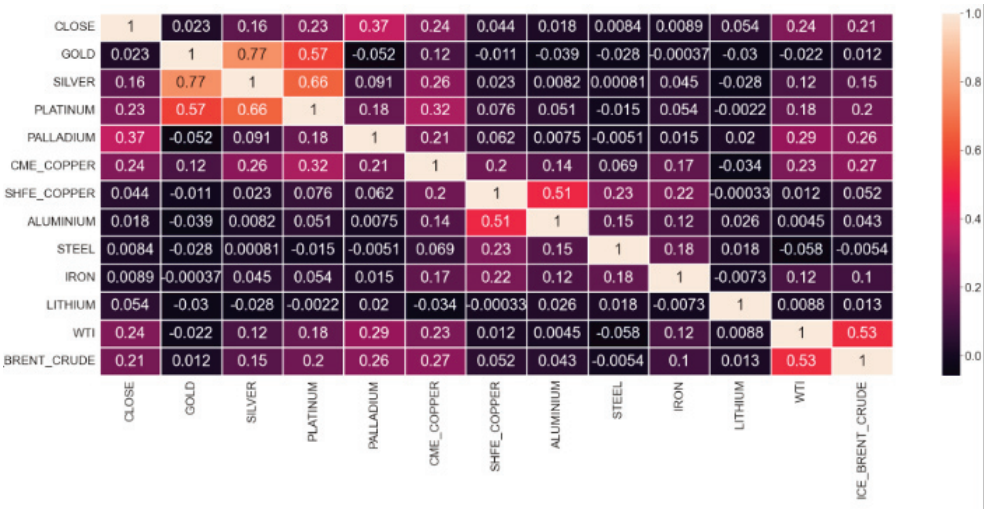
AUTOMOTIVE SECTOR - LINEAR CORRELATION COEFFICIENTS



LUXURY SECTOR - LINEAR CORRELATION COEFFICIENTS



OIL AND METAL SECTORS - LINEAR CORRELATION COEFFICIENTS



COMPANY-SPECIFIC ATTENTION WEIGHTS OF THE TEMPORAL FUSION TRANSFORMER

TECHNICAL	10%	50%	90%	FUNDAMENTAL	Mean	10%	50%	90%
Open	0,005319	0,009643	0,015057	EPS	0,015355	0,010647	0,014364	0,021624
High	0,01732	0,032128	0,042997	DY	0,013418	0,011189	0,013377	0,015877
Low	0,009699	0,012294	0,017418	PE	0,011106	0,009434	0,010925	0,012707
Close	0,008531	0,010594	0,013718	PCF	0,010761	0,006667	0,011215	0,014322
Volume	0,009005	0,015131	0,019733	ROS	**0,025125**	0,018417	0,025571	0,031913
SMA5	0,008496	0,009509	0,013976	MC	0,011205	0,007135	0,010929	0,015954
SMA10	0,007606	0,009361	0,014964	Target Price	**0,03168**	0,017216	0,035126	0,042699
SMA20	0,011809	0,015545	0,019869	Target EPS	**0,017233**	0,011094	0,016979	0,023627
EMA20	0,008781	0,011154	0,014964					
CCI	0,022396	0,030128	0,04268					
RSI	0,025332	0,039486	0,066058					

Sectorial Attention Weights of the Temporal Fusion Transformer

AUTOMOTIVE	Mean	10%	50%	90%	LUXURY	Mean	10%	50%	90%
Volkswagen	0,014989	0,009533	0,014563	0,020469	LMVH	0,020872	0,016758	0,020262	0,026352
Ford	0,011716	0,007251	0,011984	0,016116	Kering	0,008460	0,006992	0,008390	0,010231
Stellantis	0,013097	0,008744	0,013271	0,016695	Estee_Lauder	0,028096	0,016816	0,025374	0,044605
Renault	0,014962	0,012728	0,014659	0,017555	C. F. Richemont	0,018149	0,010206	0,016640	0,028348
Damler	0,016759	0,012063	0,016332	0,021240	PVH	0,012387	0,008537	0,012617	0,015867
Bayerische	0,021564	0,013307	0,021453	0,029567	Swatch	0,009966	0,005456	0,009917	0,014278
Gneral Motors	0,026570	0,02015	0,025835	0,034192	Hermes	0,015206	0,012718	0,014695	0,017268
BMW	0,011802	0,008298	0,011651	0,015772	Tapestry	0,028811	0,019475	0,026936	0,040414
Tesla	0,018313	0,013564	0,017904	0,023701	Burberry	0,040464	0,027279	0,040360	0,054307
Continental	0,013574	0,010374	0,013187	0,017343	Moncler	0,015102	0,009226	0,013706	0,021672
					Loreal	0,026146	0,016562	0,023767	0,039272
OIL&METAL	**Mean**	**10%**	**50%**	**90%**	Luxottica	0,035861	0,014767	0,033699	0,059579
Platinum	0,016795	0,012059	0,015330	0,023848	Ralph Lauren	0,013212	0,009500	0,012981	0,017219
Palladium	0,011488	0,006503	0,010758	0,017700	Capri Holding	0,012657	0,010188	0,011891	0,016043
Copper	0,026997	0,016751	0,026086	0,037885	Tiffani	0,015324	0,009621	0,014862	0,020751
WTI	0,051773	0,025413	0,052842	0,075768					
Brent	0,030536	0,020592	0,030621	0,040552					

Macroeconomic Attention Weights of the Temporal Fusion Transformer

GLOBAL&MACRO DATA	Mean	10%	50%	90%
S&P500	**0,019563**	0,010915	0,018612	0,029357
VIX	0,010892	0,009502	0,010970	0,012318
Nasdaq100	**0,019519**	0,012172	0,017997	0,029504
EuroStoxx50	**0,01721**	0,011899	0,016056	0,023659
FTSE MIB	0,016943	0,013392	0,017351	0,020250
EUR/USD	0,010796	0,007994	0,010766	0,013629
LIBOR	0,009196	0,005699	0,010236	0,011652